

Introduction to Cybersecurity, Assignment 1

Kent Odde

September 23, 2020



Contents

Abstract	3
Q1	3
Q2	4
Illustration of Vignere Ciphering	4
Vignere Cipher Program	7
Single Key	9
Two Keys	10
Q3	11
Q4 - Enigma	11
Appendices	14
Vignere Cipher Code	14
Enigma Code	17
main.cpp	17
Enigma Class	19
Rotor Class	22
References	27

Abstract

This is my submission for the first assignment in DCS3101, Introduction to Cybersecurity.

The complete source code for both the Vignere cipher and Enigma program can be found in the appendices, however I will happily provide the source files upon request.

Q1

CIA - Confidentiality, Integrity and Availability.

Alice wants to send the message "Introduction to Data and Cyber-Security (DCS3101)" to Bob. If she wants to employ the CIA concepts in the sending of this message, she has to ensure three things:

- **Confidentiality:** She will have to make sure that Bob is the only one that will be able to read the contents, and that all other third parties will not. Encrypting the message with a block encryption scheme like AES or DES, and using something like cipher-block-chaining, electronic code book etc. to handle the blocks is ways in which we Alice may obtain confidentiality.
- **Integrity:** This means making sure that Bob can be certain that the message he receives is in fact the intended message communicated by Alice. Simply put, he needs to know that the contents has not been altered in any way. Integrity can be provided by using a hashing function as a signature. If the Bob can match up the provided signature with the output of hashing part of the message, he can be sure that the message sent from Alice has not been changed in any way.
- **Availability:** All the security in the world is meaningless if Bob is not able to decrypt the message. The correct contents of the message must be available for the intended receiver, at the right time. This explanation is a bit superficial, as this topic haven't been covered as much as the others so far in the course.

These three properties are by no means independent, and one will always have to analyze ones needs, and do a trade-off analysis between the three.

High confidentiality and integrity may lead to lowered availability. Either in terms of time it takes to decrypt the message or the complexity of it. In the extreme case, the confidentiality and integrity is so high that the availability is

equal to zero. On the other hand a high emphasis on availability may compromise the level of confidentiality and integrity.

Q2

Illustration of Vignere Ciphering

In order to explain Vignere ciphering, we have to realize that Vignere ciphering is nothing more than an expanded and enhanced version of Ceasar ciphering. Let's start off by explaining a Ceasar Cipher.

Ceasar Cipher is a ciphering scheme where we apply an operation to all of the characters in a string. This may be shifting each character by +5 places. In this case we say that the key is F, as it is the letter with the value of 5, given that we count A as 0.

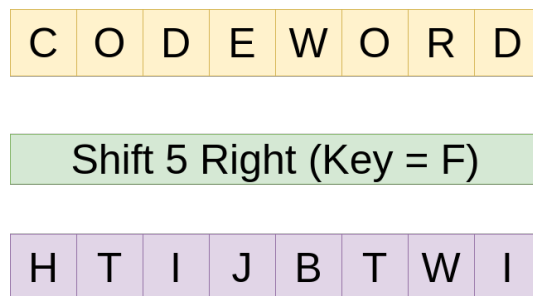


Figure 1: Simple Ceasar cipher example

There are many flaws in this scheme, where the most obvious one is that the operation is static, and if you crack one character, you crack all of them. A statistical attack can be very effective here.

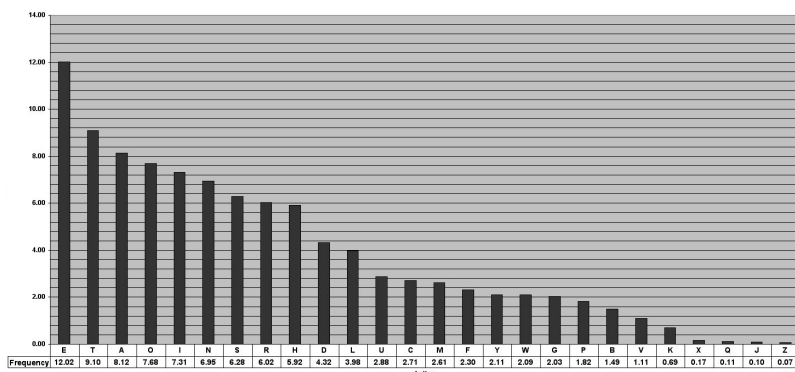


Figure 2: English letter frequency[3]

In this graph we see that the most common letters in the english language are E, T, A and O. If we assume this is also true for our ciphertex HTIJBTWI, we can guess that either the T or the I will map to one of these letters. If we start at the top with the letter I, we will only need to test 4 operations, before arriving at the correct one.

A vigner cipher attempts to increase the complexity of the Caesar Cipher by having the operations change for each character.

Instead of having a key like F in the caesarcipher, we use a key where the number of characters is greater than one. The length of the key naturally increases the robustness of the encryption.

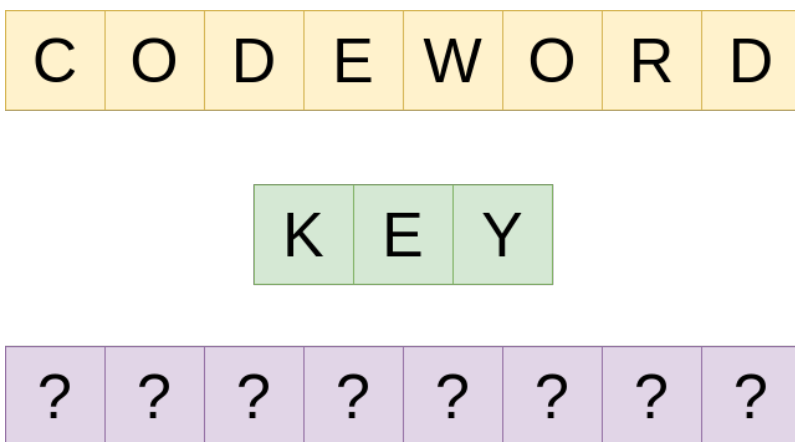


Figure 3: Vignere 1

In the example where the key is 'KEY', we use the key 'K' on the first character of the plaintext string, 'E' is the key off the second, and so on. When we reach the last character of the Key, we start over, so that the 4th character of our plaintext will also be encrypted with the key 'K'.

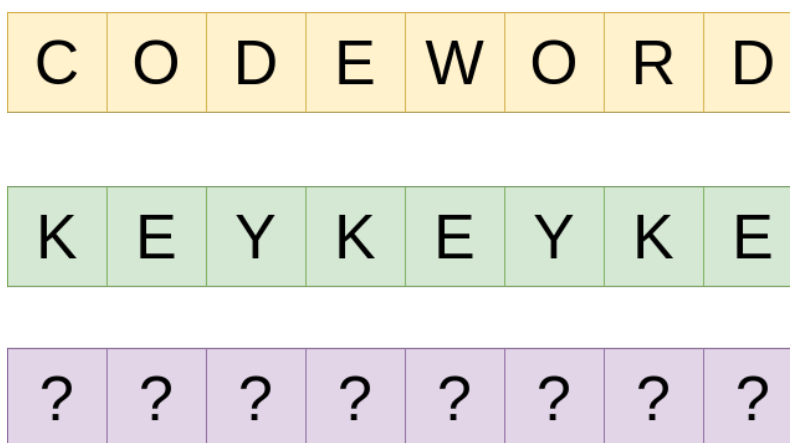


Figure 4: Vignere 2

In the following figure, we can see that the two O's and the two D's not longer map to the same letter, as was the problem with the Ceasar Cipher. In the ciphertext we can also see a couple of examples where two equal letters, like M and B, map back two different letters in the plaintext.

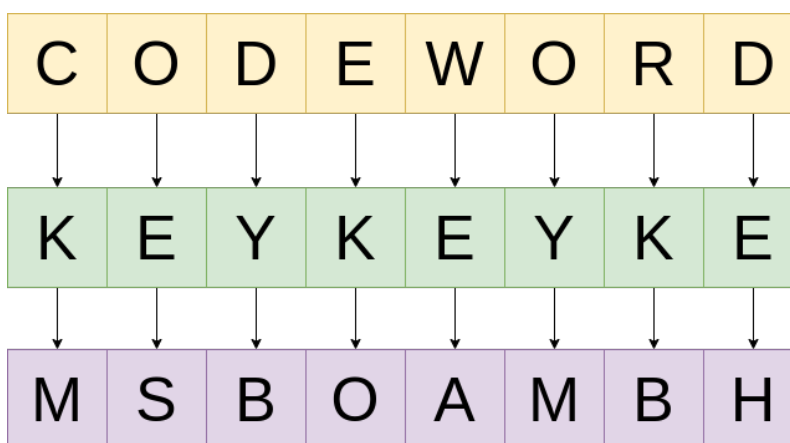


Figure 5: Vignere 3

If the message is shorter than the key, or if the key is not a multiple of the

message (like in the example), we may add padding to the plaintext before encrypting. This is a mechanism of making the messages harder to decipher, because the length of the ciphertext, does not necessarily match the length of the plaintext. The padding should not however consist of the same character. If one tries to shift the last few characters through the alphabet, the key may become obvious from this. As an illustration, let us imagine padding the message with the letter A.

H	E	L	L	O	A	A	A	A
K	E	Y	K	E	Y	K	E	Y
R	I	J	V	S	Y	K	E	Y

Figure 6: A very unfortunate padding scheme

This is of course an extreme example, because A will just give the key in plaintext, but no matter which letter we replace A with, the key will reveal itself after a maximum of 26 shift operations. Because of this, the padding scheme should include either a repetition of some part of the message, or random characters.

Vignere Cipher Program

When writing the vignere ciphering program, i assumed the following:

- Upper case characters will be encrypted within the set of upper case characters
- Lower case characters will be encrypted within the set of lower case characters
- Spaces, commas and periods will not be encrypted and only represent themselves
- Keys are of course case insensitive
- In the part with two keys, I assumed that the purpose is to encrypt the message twice, using two different keys

In the code, I found it easiest to make use of the ascii table, where upper case letters are shifted within the range of 65 and 90, and the lower case letters are in the range of 97 to 122.

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[END OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	\$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(88	58	1011000	130	X					
41	29	101001	51)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

Figure 7: ASCII table [4]

The most essence of the code lies in these two functions. We loop through the plaintext, while keeping another int to keep track of where in the keyword we are. We shift the current character in the plaintext by the relative value of the current letter in the keyword.

```
char shiftChar(char c, char shiftBy, bool backward = false)
{
    /*
    Checking if the characters upper or lower case, and chooses the
    asciiboundaries of the respective set.
    */
    int startAscii = (c >= STARTASCIILOWER) ? STARTASCIILOWER :
        STARTASCIIUPPER;
    int stopAscii = (c >= STARTASCIILOWER) ? STOPASCIILOWER :
        STOPASCIIUPPER;

    /*
    Move the character downto the 0-25 range, performing the shift, and
    moving it back up the correct ascii range.
    */
}
```



```

    */
    char output = c - startAscii;
    if(backward)
    {
        output = modulo((output - (shiftBy - STARTASCIILOWER)), (stopAscii
            - startAscii + 1)) + startAscii;
    }
    else
    {
        output = modulo((output + (shiftBy - STARTASCIILOWER)), (stopAscii
            - startAscii + 1)) + startAscii;
    }
    return output;
}

std::string encrypt(std::string plainText, std::string key)
{
    int keyIndex = 0;
    std::string cipherText;
    for(int i = 0; i < plainText.size(); i++)
    {
        /*
        Dont encrypt spaces, commas and periods.
        */
        if(plainText[i] == ' ' || plainText[i] == ',' || plainText[i] ==
            '.')
        {
            cipherText += plainText[i];
            continue;
        }
        cipherText += (shiftChar(plainText[i], key.at(keyIndex)));
        keyIndex = (keyIndex + 1) % key.size();
    }
    return cipherText;
}

```

Single Key

The text given in the task, encrypted with the key *kongsberg*, became:

Dvr wmjgb hbcjt xpb aawdf unfv kno znfq esx.

The result can also be seen in the figure below:

```
kent@kent-ThinkPad-T580: ~/Source/School/DCS3101-1_Cybersecurity/...
kent@kent-ThinkPad-T580:~/Source/School/DCS3101-1_Cybersecurity/Vignere_Cipher$ ./VignereCipher -e "The quick brown fox jumps over the lazy dog." "Kongsberg"
Input:      The quick brown fox jumps over the lazy dog.
Key:        kongsberg
Output:      Dvr wmjgb hbcjt xpb aawdf unfv kno znfq esx.
kent@kent-ThinkPad-T580:~/Source/School/DCS3101-1_Cybersecurity/Vignere_Cipher$ ./VignereCipher -d "Dvr wmjgb hbcjt xpb aawdf unfv kno znfq esx." "Kongsberg"
Input:      Dvr wmjgb hbcjt xpb aawdf unfv kno znfq esx.
Key:        kongsberg
Output:      The quick brown fox jumps over the lazy dog.
kent@kent-ThinkPad-T580:~/Source/School/DCS3101-1_Cybersecurity/Vignere_Cipher$
```

Figure 8: Vignere cipher, 1 key

Two Keys

Encrypted twice, with the keys *norway* and *oslo* respectively, became:

Ung aiyam gfzio lqh xkkrx cgqs zjo zqxa icr.

The result can also be seen in the figure below:

```
kent@kent-ThinkPad-T580: ~/Source/School/DCS3101-1_Cybersecurity/Vi...
kent@kent-ThinkPad-T580:~/Source/School/DCS3101-1_Cybersecurity/Vignere_Cipher$ ./VignereCipher -e "The quick brown fox jumps over the lazy dog." "norway" "oslo"
Input:      The quick brown fox jumps over the lazy dog.
Key:        norway
Key2:       oslo
Output:      Ung aiyam gfzio lqh xkkrx cgqs zjo zqxa icr.
kent@kent-ThinkPad-T580:~/Source/School/DCS3101-1_Cybersecurity/Vignere_Cipher$ ./VignereCipher -d "Ung aiyam gfzio lqh xkkrx cgqs zjo zqxa icr." "norway" "oslo"
Input:      Ung aiyam gfzio lqh xkkrx cgqs zjo zqxa icr.
Key:        norway
Key2:       oslo
Output:      The quick brown fox jumps over the lazy dog.
kent@kent-ThinkPad-T580:~/Source/School/DCS3101-1_Cybersecurity/Vignere_Cipher$
```

Figure 9: Vignere cipher, 2 keys

Q3

Lets start by defining the terms diffusion and confusion.

Confusion, in essence deals with obscuring the link between the key and the ciphertext, while diffusion deals with hiding the relationship between the plaintext and the ciphertext. More concretely this means that if we change a bit in the key, we must provide enough confusion, so that almost all of the ciphertext will also be affected by this change. In a similar manor, if we change a single bit of the plaintext, diffusion should be provided so that this will change at least half of the bits in the ciphertext.

In practice these two properties can be obtained in different ways. We will have a look at how they are obtained in DES (Data Encryption Standard).

DES is built of a SP-network, more specifically a feistel structure. It uses substitution and permutation over 16 rounds along with a 56 bit key, which is expanded into 16 round keys.

Confusion within DES is obtained by the substitution and the diffusion is obtained by the permutation.

Substitution is basically a rule set where in the case of DES, a six bit input, will be replaced by a four bit output, according to the specific S-box. Permutation changes the bit position of the input, which hides the statistical properties of the original plaintext.

In DES confusion is also obtained by the expansion function. It takes a 32 bit input, and produces a 48 bit output, by duplicating some of the original bits.

Q4 - Enigma

In this task, I will assume knowledge about the Enigma machine on the part of the reader, and not go into the details as this would lead to a very extensive task. There are also some conflicting sources of information about the Enigma. I have gathered all data about models, rotors and reflektors from [2] and [1]

The Enigma I, had a scheme where one would choose three rotors out of five, and one reflektor out of three. This can be seen in the figure below.

Some sources say that the Enigma I had only three rotors and some say five. However I think it was upgraded at some point, so both statements may have been true at some point. The three out of five scheme seems to be the more common one.

Wheel	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Notch	Turnover	#
ETW	ABCDEFGHIJKLMNOPQRSTUVWXYZ			
I	EKMFLGDQVZNTOWYHXUSPAIBRCJ	Y	Q	1
II	AJDKSIRUXBLHWTMCQGZNPYFVOE	M	E	1
III	BDFHJLCPRTXVZNYEIWGAKMUSQO	D	V	1
IV	ESOVZJAYQUIRXLNFTGKDCMWB	R	J	1
V	VZBRGITYUPSDNHLXAWMJQOFECK	H	Z	1
UKW-A	EJMZALYXVBWFCRQUONTSPIKHGD			
UKW-B	YRUHQLDPXNGOKMIEBFZCWVJAT			
UKW-C	FVPJIAOYEDRZXWGCTKUQSBMHL			

Figure 10: Rotor specifications for Enigma I [1]

To increase the complexity and challenge of this task, I decided implementing an M3 Enigma. This model had a total of eight rotors to choose from, instead of five. However, reflektor-A seemed to be removed. However, in order for this program to be compatible with the Enigma I, I have decided to include all three reflektors.

Wheel	ABCDEFGHIJKLMNOPQRSTUVWXYZ	Notch	Turnover	#
ETW	ABCDEFGHIJKLMNOPQRSTUVWXYZ			
I	EKMFLGDQVZNTOWYHXUSPAIBRCJ	Y	Q	1
II	AJDKSIRUXBLHWTMCQGZNPYFVOE	M	E	1
III	BDFHJLCPRTXVZNYEIWGAKMUSQO	D	V	1
IV	ESOVZJAYQUIRXLNFTGKDCMWB	R	J	1
V	VZBRGITYUPSDNHLXAWMJQOFECK	H	Z	1
VI	JPGVUUMFYQBENHZRDKASXLICTW	HU	ZM	2
VII	NZJHGRCTXMSWBOUFAIVLEPKQDT	HU	ZM	2
VIII	FKQHTLXOCBJSPDZRAMENIU YGV	HU	ZM	2
UKW-B	YRUHQLDPXNGOKMIEBFZCWVJAT			
UKW-C	FVPJIAOYEDRZXWGCTKUQSBMHL			

Figure 11: Rotor specifications for Enigma M3 [1]

For the most part this is quite a trivial task. When pushing a button, the rotors will increment. The signal will run from the button, through the plugboard, through the rotors, into the reflektor, back through the rotors, back through the plugboard, and deliver an output. In code this will for the most part consist of looking up characters in constant arrays, taking into account the current position of the rotors.

However the thing that stomped me for a while, were the ringsettings. The ringsettings are not dynamically changed, but are set by twisting the actual rotor relative to itself. This combined with the rotor position made me resort to pen and paper. Finally i was able to come up with a working function for a transformation through a rotor with a certain position and ring setting, which can be seen in the code below:

```

char Rotor::getTransformedChar(char c)
{
    //Operation: c = c + offset - ringsetting
    c = intToAsciiChar(modulo(charToAlphabetIndex(c) + offset -
        ringSetting, alphabet.size()));

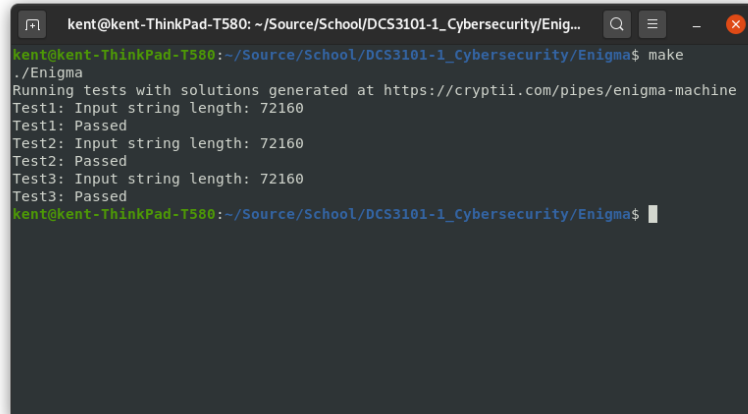
    //Operation: Normal transformation
    char o = alphabet.at(charToAlphabetIndex(c));

    //Operation: o = o + ringSetting;
    o = intToAsciiChar(modulo(charToAlphabetIndex(o) + ringSetting,
        alphabet.size()));

    //Operation: o = o - offset;
    o = intToAsciiChar(modulo(charToAlphabetIndex(o) - offset,
        alphabet.size()));
    return o;
}

```

In order to test the program properly, I generated correct ciphertexts with several different settings at <https://cryptii.com/pipes/enigma-machine>.



```

kent@kent-ThinkPad-T580: ~/Source/School/DCS3101-1_Cybersecurity/Enigma...
kent@kent-ThinkPad-T580:~/Source/School/DCS3101-1_Cybersecurity/Enigma$ make
./Enigma
Running tests with solutions generated at https://cryptii.com/pipes/enigma-machine
Test1: Input string length: 72160
Test1: Passed
Test2: Input string length: 72160
Test2: Passed
Test3: Input string length: 72160
Test3: Passed
kent@kent-ThinkPad-T580:~/Source/School/DCS3101-1_Cybersecurity/Enigma$

```

Figure 12: Test of Enigma

As can be seen in the figure, these strings were quite long in order to go through all the positions several times. These tests will be excluded from the code in the appendix, because of their size.

In the figure below, one can see a string be encrypted, decrypted and a check

performed to see that it in fact has returned the correct plaintext back.

```
./Enigma
Enigma Status:
Plugboard:
  AH MB TO EE PL KC NO ZX WR

Rotor information from left to right:

L-rotor
  RotorId: 4
  Stepping point: 10
  Current offset:23
  Current RingSetting: 12
  Alphabet: ESOVPZJAYOUIRXNLNFTGKDCQMB

M-rotor
  RotorId: 5
  Stepping point: 0
  Current offset:11
  Current RingSetting: 1
  Alphabet: VZMRGUTYUPOHNLXAMJQDFECK

R-rotor
  RotorId: 6
  Stepping point: 0
  Stepping point2: 13
  Current offset:25
  Current RingSetting: 20
  Alphabet: JPOVUMFYOBENHZDKASLICTW

Reflector: 2
  Alphabet: YRUHQSLDPXNGOMNIEBZCRUJAT

Input:  THISC OURSE ISANI NTRDO UCTIO NTOCY BERSE CURIT YTHIS COURS EISAN INTRD OUCTI ONTOC YBERS ECURI TYTHI SCOUR SEISA NINTR ODUCT IONTO CYBER SECUR ITY
Encrypted: SPXDH DNZQD NANKE UIVRS OYLZX EZLHN VCKOJ BLJXS HPWSA FFWOD KCAUK ZHHTT ZEUCB YHMID NLQUX JPJFM UPPIT HPUGL GTCUD RGSFD PKGFS QAJFN SFQOB ILZCH XPU
Decrypted: THISC OURSE ISANI NTRDO UCTIO NTOCY BERSE CURIT YTHIS COURS EISAN INTRD OUCTI ONTOC YBERS ECURI TYTHI SCOUR SEISA NINTR ODUCT IONTO CYBER SECUR ITY
Checking correct decryption of 123 characters...
Decrypt Success
```

Figure 13: Enigma

Appendices

Vignere Cipher Code

```
#include <iostream>
#include <string>
#include <algorithm>

int STARTASCIILOWER = 97;
int STOPASCIILOWER = 122;
int STARTASCIUPPER = 65;
int STOPASCIUPPER = 90;

/*
As the %-operator does not behave like modulo on negative numbers, this
function is necessary
*/
int modulo(int a, int b)
{
    return (a % b + b) % b;
}

char shiftChar(char c, char shiftBy, bool backward = false)
{
    /*
```

```

Checking if the characters upper or lower case, and chooses the
    asciiboundaries of the respective set.
*/
int startAscii = (c >= STARTASCIILOWER) ? STARTASCIILOWER :
    STARTASCIIUPPER;
int stopAscii = (c >= STARTASCIILOWER) ? STOPASCIILOWER :
    STOPASCIIUPPER;

/*
Move the character down to the 0-25 range, performing the shift, and
    moving it back up the correct asciirange.
*/
char output = c - startAscii;
if(backward)
{
    output = modulo((output - (shiftBy - STARTASCIILOWER)), (stopAscii
        - startAscii + 1)) + startAscii;
}
else
{
    output = modulo((output + (shiftBy - STARTASCIILOWER)), (stopAscii
        - startAscii + 1)) + startAscii;
}
return output;
}

std::string encrypt(std::string plainText, std::string key)
{
    int keyIndex = 0;
    std::string cipherText;
    for(int i = 0; i < plainText.size(); i++)
    {
        /*
        Dont encrypt spaces, commas and periods.
        */
        if(plainText[i] == ' ' || plainText[i] == ',' || plainText[i] ==
            '.')
        {
            cipherText += plainText[i];
            continue;
        }
        cipherText += (shiftChar(plainText[i], key.at(keyIndex)));
        keyIndex = (keyIndex + 1) % key.size();
    }
    return cipherText;
}

std::string decrypt(std::string cipherText, std::string key)
{
    int keyIndex = 0;

```

```

std::string plainText;
for(int i = 0; i < cipherText.size(); i++)
{
    /*
    Dont decrypt spaces, commas and periods.
    */
    if(cipherText[i] == ' ' || cipherText[i] == ',' || cipherText[i]
        == '.')
    {
        plainText += cipherText[i];
        continue;
    }

    plainText += (shiftChar(cipherText[i], key.at(keyIndex), true));
    keyIndex = (keyIndex + 1) % key.size();
}
return plainText;
}

int main(int argc, char* argv[])
{
    /*
    Mostly commandline argument handling
    */
    if(argc < 4)
    {
        std::cout << "Wrong number of arguments passed\n";
        std::cout << "Arguments: \n\t./Vignere cipher -e/d <string> <key>
            <optionalSecondKey>\n";
        return -1;
    }

    std::string input = argv[2];
    std::string key = argv[3];
    std::transform(key.begin(), key.end(), key.begin(), ::tolower);
    std::string key2 = "";
    if(argc > 4)
    {
        key2 = argv[4];
        std::transform(key2.begin(), key2.end(), key2.begin(), ::tolower);
    }
    std::string output;

    if(std::string(argv[1]) == "-e")
    {
        output = encrypt(input, key);

        if(key2.compare("") != 0)
        {

```



```

        output = encrypt(output, key2);
    }
}
else if(std::string(argv[1]) == "-d")
{
    output = decrypt(input, key);

    if(key2.compare("") != 0)
    {
        output = decrypt(output, key2);
    }
}
else
{
    std::cout << "Error, expected arguments:\n\t/Vignere cipher -e/d
        <string> <key> <optionalSecondKey>\n";

    return -1;
}
std::cout << "\nInput:\t\t" << input << "\n";
std::cout << "Key:\t\t" << key << "\n";
if(key2.compare("") != 0) std::cout << "Key2:\t\t" << key2 << "\n";
std::cout << "Output:\t\t" << output << "\n";
return 0;
}

```

Enigma Code

main.cpp

```

#include <string>
#include <iostream>
#include "Rotor.h"
#include "Enigma.h"

std::string formatString(std::string s);

int main(int argc, const char* argv[])
{
    //Setup Enigma
    //Choose rotors 1-8 - Left to right;
    int rotorIds[3] = {4,5,6};
    //Choose Reflektor 1-3 (ABC)
    int reflektorId = 2;
    //Set offset and ringSetting 1-26 - Left to right;
    int offset[3] = {24,12,26};
}

```



```
    return s;
}
```

Enigma Class

Enigma.h

```
#ifndef ENIGMA_H
#define ENIGMA_H
#include "Rotor.h"
#include <vector>
#include <algorithm>

class Enigma
{
public:
    Enigma(int rotorIds[3], int reflectorId, std::vector<std::string>
        _plugboardSettings);
    ~Enigma();
    void printEnigmaStatus();
    void setRotors(int r[3]);
    void setOffset(int o[3]);
    void setRingSetting(int s[3]);
    std::string transform(std::string& input);

private:
    Rotor reflector;
    std::vector<Rotor> rotors;
    std::vector<std::string> plugboardSettings;
    void rotateRotors();
    char charThroughPlugboard(char c);
};
#endif
```

Enigma.cpp

```
#include "Enigma.h"

Enigma::Enigma(int rotorIds[3], int reflectorId,
    std::vector<std::string> _plugboardSettings):
    reflector(reflectorId, 'r')
{
    Rotor rotor1(rotorIds[0], 'L');
    Rotor rotor2(rotorIds[1], 'M');
    Rotor rotor3(rotorIds[2], 'R');
    rotors.push_back(rotor1);
```

```

        rotors.push_back(rotor2);
        rotors.push_back(rotor3);
        plugboardSettings = _plugboardSettings;
    }

    Enigma::~Enigma()
    {

    }

    void Enigma::printEnigmaStatus()
    {
        std::cout << "Enigma Status: \n\tPlugboard: \n\t";
        for(auto pair : plugboardSettings)
        {
            std::cout << pair << " ";
        }

        std::cout << "\n\nRotor information from left to right: \n\n";

        for(auto rotor : rotors)
        {
            rotor.printRotorStatus();
        }

        reflector.printRotorStatus();
    }

    void Enigma::setRotors(int r[3])
    {
        for(int i = 0; i <= 2; i++)
        {
            Rotor rotor(r[i], r[i]);
            rotors.push_back(rotor);
        }
    }

    void Enigma::setOffset(int o[3])
    {
        for(int i = 0; i < rotors.size(); i++)
        {
            rotors.at(i).setOffset(o[i] - 1);
        }
    }

    void Enigma::setRingSetting(int s[3])
    {
        for(int i = 0; i < rotors.size(); i++)

```

```

    {
        rotors.at(i).setRingSetting(s[i] - 1);
    }
}

void Enigma::rotateRotors()
{
    for(auto it = rotors.rbegin(); it != rotors.rend(); it++)
    {
        bool next = it->incrementOffset(rotors.at(1).isDoubleStep());
        if(!next) break;
    }
    //printRotorStatus();
}

std::string Enigma::transform(std::string& input)
{
    std::string output;

    for(const char c : input)
    {
        rotateRotors();

        char temp = std::toupper(c);
        //Through the Plugboard
        temp = charThroughPlugboard(temp);

        //Through the rotors
        for(auto it = rotors.rbegin(); it != rotors.rend(); it++)
        {
            temp = it->getTransformedChar(temp);
        }
        //Through the reflector
        temp = reflector.getTransformedChar(temp);

        //Through the rotors backwards
        for(auto it = rotors.begin(); it != rotors.end(); it++)
        {
            temp = it->getInverseTransformedChar(temp);
        }

        temp = charThroughPlugboard(temp);

        output += temp;
    }
    return output;
}

```

```

char Enigma::charThroughPlugboard(char c)
{
    for(auto pair : plugboardSettings)
    {
        if(pair.front() == c) return pair.back();
        if(pair.back() == c) return pair.front();
    }
    return c;
}

```

Rotor Class

Rotor.h

```

#include <string>
#include <iostream>
#ifndef ROTOR_H
#define ROTOR_H

class Rotor
{
public:
    Rotor();
    Rotor(int _id, char _pos);
    ~Rotor();
    char getTransformedChar(char c);
    char getInverseTransformedChar(char c);
    void setRingSetting(int i);
    bool incrementOffset(bool doubleStep);
    void setOffset(int i);
    void printRotorStatus();
    bool isDoubleStep();
private:
    std::string alphabet;
    int ringSetting;
    int offset;
    int steppingPoint;
    int steppingPoint2 = -1;
    char intToAsciiChar(int i);
    int rotorId = 0;
    char rotorPosition;
    int charToAlphabetIndex(char c);
    int modulo(const int& a, const int& b);
    const int asciiOffset = 65;
};

//Assorted rotor data etc.

```

```

static const std::string masterAlphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
static const std::string rotorAlphabets[9] =
{
    "", //Dummy alphabet
    "EKMFLGDQVZNTOWYHXUSPAIBRCJ",
    "AJDKSIRUXBLHWTMCQGZNPYFVOE",
    "BDFHJLCPRTXVZNYEIWGAKMUSQO",
    "ESOVZPJAYQUIRHXNLFTGKDCMWB",
    "VZBRGITYUPSDNHLXAWMJQOFECK",
    "JPGVOUMFYQBENHZRDKASXLICTW",
    "NZJHGRCXMYSWBOUFAIVLPEKQDT",
    "FKQHTLXOCBJSPDZRAEWNUIUYGV"
};
//Rotors will step the next, when stepping to these characters
static const std::string stepPoints[] =
{
    "O", //Dummy stepper
    "R",
    "F",
    "W",
    "K",
    "A",
    "AN" //Have to add another stepping point at 'N' for this
};

static const std::string reflectorAlphabets[] =
{
    "", //Dummy alphabet
    "EJMZALYXVBWFCRQUONTSP IKHGD",
    "YRUHQSLDPXNGOKMIEBFZCWVJAT",
    "FVPJIAOYEDRZXWGCTKUQSBNMHL"
};
#endif

```

Rotor.cpp

```

#include "Rotor.h"

Rotor::Rotor()
{

}

Rotor::Rotor(int _id, char _pos)
{
    //A bit of extra overhead here, because a rotor may be a reflektor, or
    it may have two steppingpoints

    rotorPosition = _pos;
}

```

```

    if(_pos == 'r')
    {
        rotorId = _id;
        alphabet = reflectorAlphabets[_id];
    }
    else
    {
        rotorId = _id;
        alphabet = rotorAlphabets[_id];
    }
    ringSetting = 0;
    offset = 0;

    if(_id == 6 || _id == 7 || _id == 8)
    {
        steppingPoint = charToAlphabetIndex(stepPoints[6].at(0));
        steppingPoint2 = charToAlphabetIndex(stepPoints[6].at(1));
    }
    else
    {
        steppingPoint = charToAlphabetIndex(stepPoints[_id].at(0));
    }
}

Rotor::~Rotor()
{
}

int Rotor::modulo(const int& a, const int& b)
{
    return (b + (a % b)) % b;
}

char Rotor::getTransformedChar(char c)
{
    //Operation: c = c + offset - ringsetting
    c = intToAsciiChar(modulo(charToAlphabetIndex(c) + offset -
        ringSetting, alphabet.size()));

    //Operation: Normal transformation
    char o = alphabet.at(charToAlphabetIndex(c));

    //Operation: o = o + ringSetting;
    o = intToAsciiChar(modulo(charToAlphabetIndex(o) + ringSetting,
        alphabet.size()));

    //Operation: o = o - offset;
    o = intToAsciiChar(modulo(charToAlphabetIndex(o) - offset,
        alphabet.size()));
}

```



```

        return o;
    }

    char Rotor::getInverseTransformedChar(char c)
    {
        //Operation: c = c + offset;
        c = intToAsciiChar(modulo(charToAlphabetIndex(c)+offset,
            alphabet.size()));

        //Operation: c = c -ringSetting;
        c = intToAsciiChar(modulo(charToAlphabetIndex(c) - ringSetting,
            alphabet.size()));

        //Operation: What gives c in masteralphabet:
        int i = alphabet.find(c);
        char o = masterAlphabet.at(i);

        //Operation: o = o - offset + ring
        o = intToAsciiChar(modulo(charToAlphabetIndex(o) - offset +
            ringSetting, alphabet.size()));

        return o;
    }

    void Rotor::setRingSetting(int i)
    {
        ringSetting = i;
    }

    bool Rotor::incrementOffset(bool doubleStep)
    {
        offset = modulo(offset + 1, alphabet.size());
        //If this is the rightmost rotor, and the middle rotor is in
        //doublestep position, we also have to return true;
        if((offset == steppingPoint || offset == steppingPoint2) ||
            rotorPosition == 'R' && doubleStep)
        {
            return true;
        }
        return false;
    }

    void Rotor::setOffset(int i)
    {
        offset = i;
    }

    char Rotor::intToAsciiChar(int i)
    {
        return char(i + asciiOffset);
    }

```

```

}

int Rotor::charToAlphabetIndex(char c)
{
    int i = (int)c - asciiOffset;
    return i;
}

void Rotor::printRotorStatus()
{
    if(rotorPosition == 'r')
    {
        std::cout << "\tReflector: " << rotorId << "\n";
        std::cout << "\t\tAlphabet: " << alphabet << "\n\n";
    }
    else
    {
        std::cout << "\t" << rotorPosition << "-rotor\n";
        std::cout << "\t\tRotorId: " << rotorId << "\n";
        std::cout << "\t\tStepping point: " << steppingPoint << "\n";
        if(steppingPoint2 != -1) std::cout << "\t\tStepping point2: " <<
            steppingPoint2 << "\n";
        std::cout << "\t\tCurrent offset: " << offset << "\n";
        std::cout << "\t\tCurrent RingSetting: " << ringSetting << "\n";
        std::cout << "\t\tAlphabet: " << alphabet << "\n\n";
    }
}

//If the middle rotor is at the position before it is supposed to step
//the next, it shall just step to the next;
bool Rotor::isDoubleStep()
{
    bool ret = (rotorPosition == 'M' && (offset == modulo(steppingPoint -
        1, alphabet.size()) || (steppingPoint2 != -1 && offset ==
        modulo(steppingPoint2 - 1, alphabet.size()))));
    return ret;
}

```

References

- [1] CryptoMuseum.com, enigma. <https://www.cryptomuseum.com/crypto/enigma/wiring.htm#7>. Accessed: Sept-20.
- [2] Dirk Rijmenants, technical details of the enigma machine. <http://users.telenet.be/d.rijmenants/en/enigmatech.htm>. Accessed: Sept-20.
- [3] English Letter Frequency, cornell university. <http://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html>. Accessed: Sept-20.
- [4] Wikimedia Commons, ascii-table. <https://commons.wikimedia.org/wiki/File:ASCII-Table.svg>. Accessed: Sept-20.